VORTEXJE - AN OPEN-SOURCE PANEL METHOD FOR CO-SIMULATION

JORN H. BAAYEN

Abstract. This paper discusses the use of the panel method for dynamical system simulation. Specifically, the advantages and disadvantages of model exchange versus co-simulation are discussed. Based on a trade-off analysis, a set of recommendations for a panel method implementation and for a co-simulation environment is proposed. These recommendations are implemented in a C++ library, offered on-line under an open source license by the author. This code is validated against XFLR5, and its suitability for co-simulation is demonstrated with an example of a tethered wing, i.e, a kite. The panel method implementation and the co-simulation environment are shown to be able to solve this stiff problem in a stable fashion.

1. Introduction

This paper focuses on the application of the panel method [14, 19, 7, 17] to simulation of dynamical systems interacting with fluid flow. The primary advantage of a potential flow method over the Navier-Stokes equations is the ease by which they can be solved numerically. A panel method has unknowns on the object surface only, whereas numerical solution of the Navier-Stokes equations requires a 3-dimensional mesh throughout the flow region. This reduced computational requirement renders the panel method a viable candidate for dynamic simulation.

Our aim lies in the development of an open-source implementation of the panel method specifically designed for dynamic simulation, where the aerodynamic forces and moments interact with the system dynamics.

An example of such a system is a wing tethered to the ground, i.e, a kite. We use the panel method to obtain the aerodynamic forces and moments acting on the kite, and an ODE solver to compute the shape and tension of the tether. The aerodynamic forces of the kite act on the tether, and the other way around. We use this stiff interaction as a test problem for the development of a panel method co-simulation methodology.

2. Existing Open Panel Method Implementations

The XFOIL package [11] and its derivate XFLR5 [8] offer an open-source implementation of the source-doublet panel method described in [19]. Both share a tight integration between the method and a user interface designed for fixed flight condition analysis, rendering them unpractical for dynamic simulation.

While it may have been possible to create a package capable of dynamic simulation by reassembling parts of these existing implementations, we chose to start from a clean slate. In this way we were able to ensure that the code follows our

Date: 19/10/12.

design goals, instead of the other way around. Furthermore, it has been our aim to design the implementation such that graphical interfaces could readily be built using our implementation as a reusable component.

3. Use Case: A Tethered Wing

In this section we introduce a test problem for verification of our design decisions.

During the last two decades, kites, i.e. tethered wings, have gained popularity not only for sports, but also for the towing of ships [20], and for the generation of wind energy on land [18, 6]. In our case we investigate the kite as it poses a stiff co-simulation problem, and thereby a solid challenge for our simulation paradigm.

We model the kite aerodynamics using the panel method, and the tether as a string of point-masses connected by high-modulus spring-dampers, represented by an ODE. The aerodynamic forces acting on the kite, resulting from the panel method, are added to the spring-damper forces of the top point-mass in the tether model – see Figure 3.1. In this way the bulk of the kite's lift force is balanced by the line forces. As a consequence, the aerodynamic forces and the kinematics operate on different orders of magnitude, resulting in a stiff interaction.

We generated a mesh for a typical C-shaped surf kite with the geometry specified in Table 1. The resulting mesh is shown in Figure 3.1.

Parameter	Value
Airfoil	Clark-Y
Aspect ratio	6.0
Projected aspect ratio	4.5
Tip alignment	Trailing edge
Root chord	3 m
Tip/chord ratio	0.25
Airfoil panels	18
Spanwise panels	18

Table 1. Kite parameters

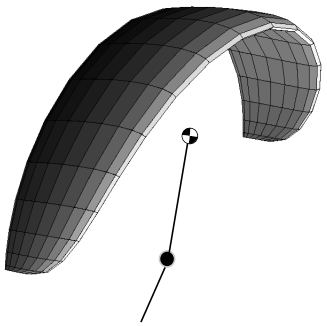

Figure 3.1. Kite and tether co-simulation.

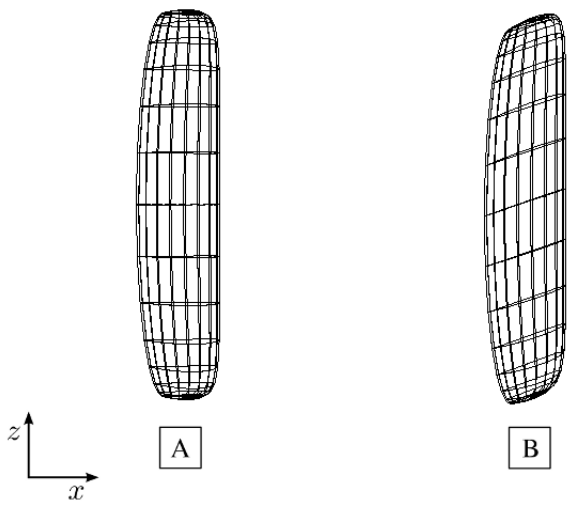

FIGURE 3.2. Kite mesh without steering deformation (A) and with steering deformation $u = 1$ (B). Top view.

Such surf kites yaw in response to asymmetrical deformations [5]. To obtain such asymmetrical states, we set up the following dynamic deformation in the spanwise coordinate:

$$\Delta z = ux,$$

where x is a dimensionless chordwise coordinate between 0 and 1. The steering input u is commanded as a single scalar input variable. The top view of an undeformed mesh, and of a deformed mesh with a steering input of $u = 1$, is displayed in Figure 3.2.

Further deformation as well as viscuous drag was neglected. While viscuous drag is crucial in obtaining an accurate model, in this work we are primarily interested in obtaining a rudimentary model suitable for testing our co-simulation setup.

The equations of the tether model are trivial but space-consuming. For this reason, we have deferred them to Appendix A.

4. Model Exchange versus Co-Simulation

In the present section we discuss two different practices that can be used to integrate the panel method with a dynamical system simulator. Of primary importance is the distinction between the model exchange and co-simulation paradigms [4].

Model exchange refers to the practice of wrapping an external model in a stateless function, evaluated as part of an ODE or DAE. The external model thereby becomes a part of a containing dynamical system model. In our case, any states such as the source and doublet distributions, would become part of the system of equations. We note that these states must indeed be stored, as a proper use of the unsteady Bernoulli equation does require knowledge of the time-variation of the source and doublet distributions. The disadvantage of model exchange for the panel method is that the size of the system matrix grows quadratically in the number of panels.

Furthermore, all but the most trivial solvers would make repeated calls to the panel method function. This would pose a performance problem due to the overhead of solving the – usually large – linear system for computing the doublet distribution.

A co-simulation master juxtaposes two simulation environments, each with their own internal states. The master coordinates interaction between the simulation environments at pre-defined events or times, and the simulators are commanded to step forward in turns [4]. In a co-simulation environment, a panel method simulator is able to maintain internal states without expanding the system matrices of the dynamical system simulator.

Stiff interactions between the panel method and the dynamical system simulator pose a challange for both concepts. In a model exchange environment, performance of the overall system solver would be limited by an implicit solver evaluating the panel method Jacobian using finite differences. The same holds for co-simulation context, where in this case a specialized implicit co-simulation solver would need to be developed in addition.

Comparing our two options we see that both suffer from the panel method's linear system overhead equally. Co-simulation, however, does not suffer from quadratic growth of the system matrix. This advantage compels us to focus on the co-simulation paradigm.

5. Panel Method API Design

As our focus lies with dynamic simulation, rather than with highly detailed aerodynamic analysis, we chose to implement a classical first-order source-doublet panel method as described in [18]. For computation of the surface velocities we use N. Marcov's formula [10], and for 3-D mesh generation we follow the algorithm detailed in [7].

Furthermore, during a dynamic simulation a body may find itself interacting with its own wake. Therefore we supplemented the classical method with singularity elimination following Dixon [9].

Aeroelastacity and boundary layer models were not considered, but may be implemented in the future.

We will now briefly discuss the considerations resulting in our software architecture. In order to facilitate a modular description of an object in flow, we chose to allow for a multitude of meshes to be specified. Wake-emitting, wake, and source-only meshes are allowed. In line with the common approach of modeling an airplane as consisting of wake-emitting and source-only parts [7, 17], we chose to implement an additional class for collecting a multitude of meshes. In this way, the programmer may handle the airplane as one, applying transformations to the collection as if it were a single object.

These design considerations are encoded in the diagram shown in Figure 5. The classes shown in this diagram are:

- `Mesh`. This object implements file input/output to and from the Gmsh [12] MSH file format, representing a mesh using a variant of the face-vertex representation [22]:
  - Vertex geometry: Vertex ID $\to (x, y, z)$;
  - Panel vertices: Panel ID $\to$ (Node ID 1, Node ID 2, ...);
  - Panels with a shared edge: Panel ID $\to$ (Panel ID 1, Panel ID 2, ...);
  - Vertex neighbors: Vertex ID $\to$ (Panel ID 1, Panel ID 2, ...).

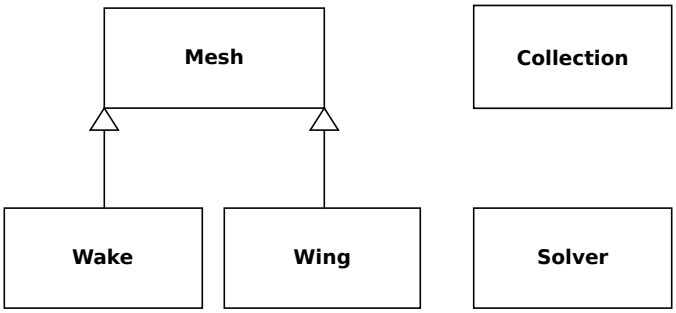

FIGURE 5.1. Class diagram.

- `Wake`. This subclass of `Mesh` represents a wake using a doublet-only vortex sheet. It contains code to emit new wake panels.
- `Wing`. This subclass of `Mesh` represents a wake-emitting wing, and contains code for managing the trailing edge.
- `Collection`. This is a container object for grouping meshes and wings together, for instance to represent an aircraft with wake-emitting parts and with source-only parts. It contains code for applying affine operations to a whole collection at once.
- `Solver`. This object contains an implementation of the actual panel method. It uses the BI-CGSTAB [23] implementation from EIGEN [13] to solve the doublet coefficient equations. The coefficient distributions from the previous timestamp are cached for use by the unsteady Bernoulli equation to compute the pressure distributions.

The listed classes were implemented in C++ in a library called VORTEXJE, available on-line [2] under the terms of an open-source license.

In the next section we will compare simulation results from VORTEXJE and XFLR5 for a simple example, before, in the section thereafter, embarking on a co-simulation of a kite with its tether.

6. VALIDATION

To validate our implementation, we simulated a 3-D wing with the properties specified in Table 2 using both XFLR5 version 6.07 in inviscid mode, and VORTEXJE. The pressure distribution of this wing, computed by VORTEXJE at an angle of attack of 10 degrees, is shown in Figure 6.1. A comparison of the integrated pressure distributions of XFLR5 and VORTEXJE is shown in Figure 6.2.

On closer inspection of the subtle differences shown in Figure 6.2, we found that XFLR5 computes the potential gradient on the wing surface as the gradient of the doublet distribution:

$$\vec{v}(\vec{x}) = -(\nabla\mu)(\vec{x}).$$

This assumption originates from [19], where it is stated without proof. We, on other hand, implemented N. Marcov's formula [10], which derives an alternative expression for the surface-derivative of the potential:

Parameter	Value
Airfoil	NACA0012
Span	6 m
Chord	1 m
Airfoil panels	32
Spanwise panels	40
Airspeed	30 m/s
Fluid density	1.2 kg/m^3

TABLE 2. Validation set-up.

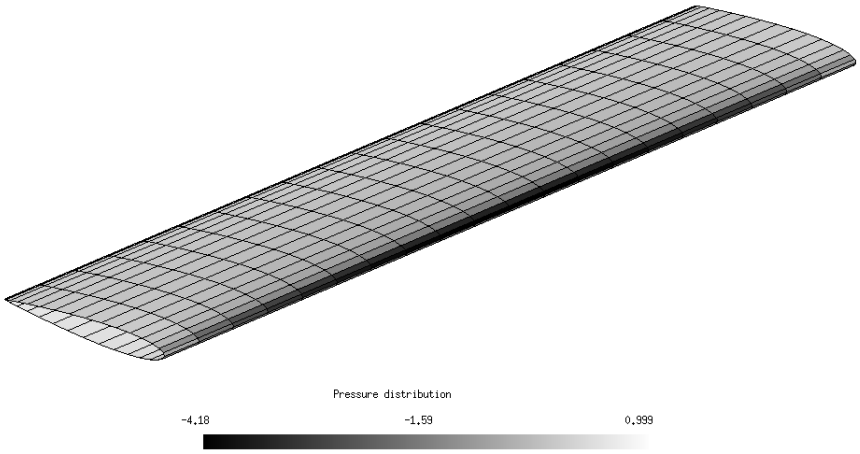

FIGURE 6.1. Pressure distribution, as computed by VORTEXJE.

$$\vec{v}(\vec{x}) = \vec{w}(\vec{x}) - \frac{1}{2}(\nabla\mu)(\vec{x}), \tag{6.1}$$

with the term $\vec{w}(\vec{x})$ representing the Cauchy principal value of the potential gradient contribution of the entire boundary. To compute this value numerically, we sum up the potential gradients of all panels, treating the local panel the same as all others. We do not run into the theoretical singularity this way, as the explicit expressions of the potential gradients for source and doublet panels are singular only on panel boundaries [17]. The *theoretical* singularity of the local potential contribution at the point of evaluation, however, is significant, and gives rise to the second term in Equation 6.1.

Apart from these differing approaches to the computation of the surface velocity, differences in numerical implementation – such as our omission of the far-field formulas [17] – will also be responsible for the results of Figure 6.2.

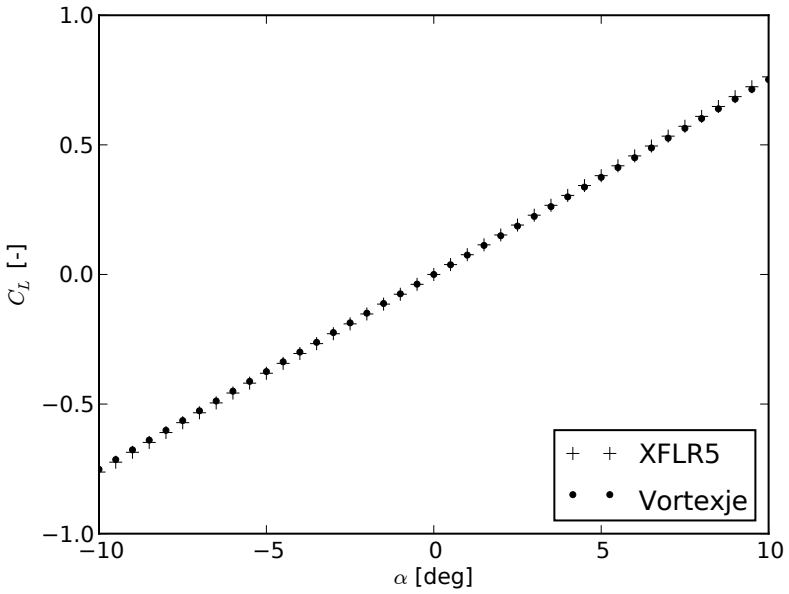

FIGURE 6.2. Comparison of XFLR5 and VORTEXJE simulation results.

7. CO-SIMULATION METHODOLOGY

We will now return to co-simulation, and verify our approach with our case problem, the tethered kite.

For encoding the tether ODE, we selected MODELICA for its ease of use and extensibility. As the bulk of the computational time will be spent in the ODE and panel method solvers, we chose to implement the co-simulation master in the PYTHON scripting language. The JMODELICA project [1] maintains an open-source Python package for simulating model exchange FMUs, PYFMI [4].

We also used JMODELICA to compile the tether model to an FMU, and used PYFMI to access the FMU from Python. JMODELICA's ASSIMULO package – which wraps the SUNDIALS suite of integrators [15] – was used to integrate the tether model.

On the upper end of the tether, we implemented a kite mesh with predefined deformation states in C++. As of this writing, PYFMI does not support the FMI co-simulation standard. Therefore we used a BOOST.PYTHON [21] class to set up communication between PYTHON and VORTEXJE instead. Figure 7.1 summarizes the complete set-up.

Both simulators are stepped in turn. The step size is reduced on a binary logarithmic scale until roughly linear behaviour is achieved:

$$\frac{|\|y(t+h) - y(t)\| - \|\dot{y}\|h|}{\|\dot{y}\|h} < \varepsilon,$$

where ε is a pre-defined linearity tolerance value. The step size reduction is carried out down to a minimum step size h_0. Every T_h time steps the step size

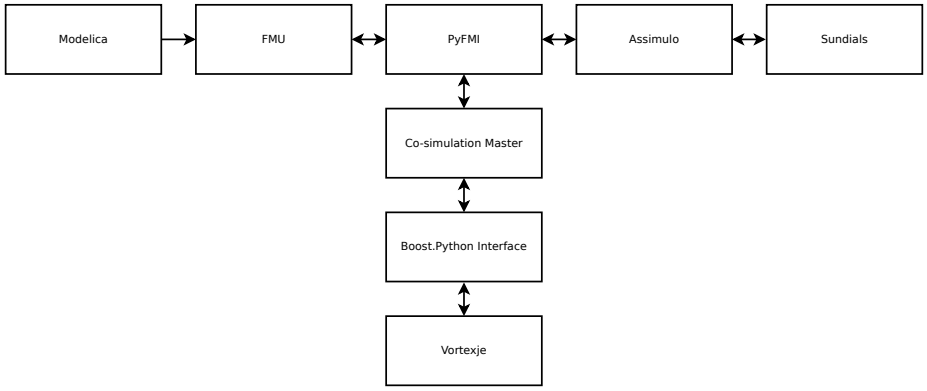

FIGURE 7.1. Co-simulation architecture.

is doubled, up to a maximum step size h_1, before entering the step size reduction algorithm. In this way, we ensure that the simulator does not remain stuck with a very small step size after integrating an interval of high stiffness, while at the same time refraining from trying an excessively large step size in vain too often. This stepping scheme is summarized as a flow chart in Figure 7.2. The parameters values selected for the solver are shown in Table 3.

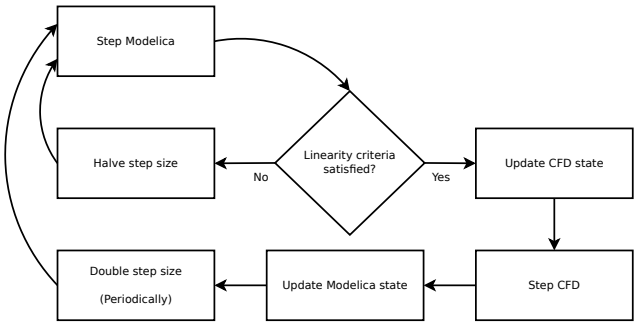

FIGURE 7.2. Co-simulation flow chart.

With the control algorithm from [3], the trajectory of the kite is shown in Figure 7.3. No numerical instability was observed.

8. Conclusions and Future Directions

In this work we proposed a design for a panel method implementation capable of efficient co-simulation. The design was implemented in a C++ code, released on-line under an open-source license. We continued with validation results, followed by an analysis of a co-simulation of a kite with its tether. This tether is composed of point-masses and spring-dampers, and the interaction of the spring-damper and aerodynamic kite forces results in a stiff problem. We showed that our co-simulation framework – implemented entirely using open-source software – is able to solve the kite problem in a stable fashion.

Setting	Value
Minimum master step size h_0	1×10^{-6}
Maximum master step size h_1	5×10^{-3}
Step size doubling period T_h	10
Linearity tolerance ε	0.1
SUNDIALS solver	CVODE
CVODE absolute tolerance	1×10^{-6}
CVODE relative tolerance	1×10^{-6}
Bi-CGSTAB maximum iterations	20×10^{3}
Bi-CGSTAB tolerance	1×10^{-10}

TABLE 3. Solver parameters.

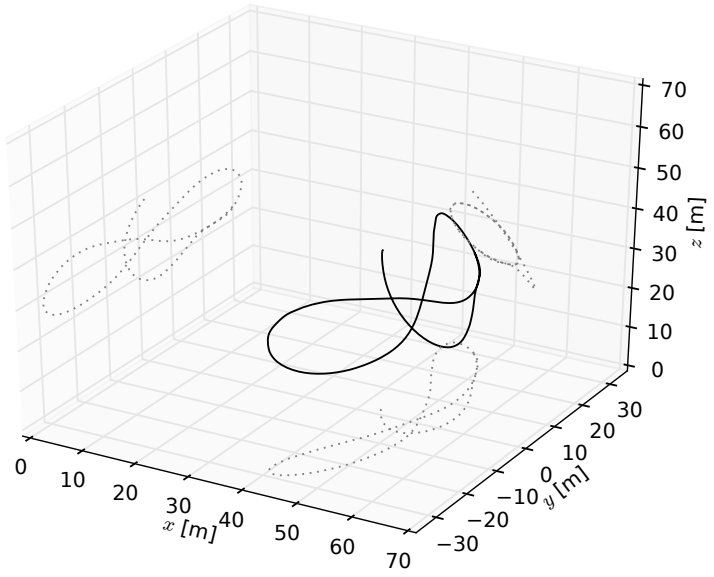

FIGURE 7.3. Co-simulated kite trajectory.

Future work will focus on adding optional aeroelasticity and boundary layer simulation features to the software. We invite interested readers to evaluate our software, and hope to establish a community of users and developers.

APPENDIX A: TETHER MODEL

We model the flexible tether by connecting a string of point-masses and spring-dampers. The springs generate aerodynamic drag force proportional to their current length, and this force is distributed over the neighboring point-masses. All point-masses have gravity acting on them.

To make this precise, we fix a spring and describe the forces acting on its neighbouring point-mass nodes. Let us assume a fixed spring and let $\vec{d}$ be a unit vector pointing from the top incident mass to the bottom incident mass or the other way

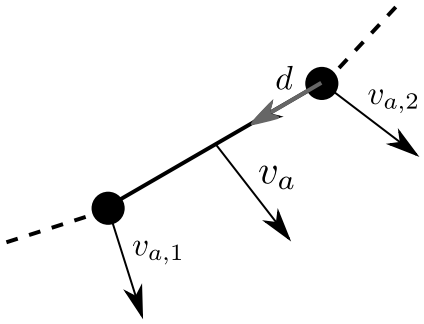

FIGURE 8.1. A spring-damper and incident masses.

around. Let $\vec{v}_a$ be the average aerodynamic velocity of the incident masses – see Figure 8 – and let

$$\vec{v}_{a,\perp} := \vec{v}_a - (\vec{v}_a \cdot \vec{d})\vec{d},$$

the component of $\vec{v}_a$ that is perpendicular to $\vec{d}$. We model the drag force as acting in the opposite direction of $\vec{v}_{a,\perp}$ with magnitude

$$F_a = \frac{1}{2}\rho \|\vec{v}_{a,\perp}\|^2 A C_D$$

with cable element surface area A and a dimensionless drag coefficient C_D.

To calculate tether forces, one can fix either incident node and let $\vec{d}$ point towards the mass in question. The spring force acts along the vector $\vec{d}$, and is given by a one-sided version of Hooke's law

$$F_s = \begin{cases} -k(l - l_0) & \text{if } l > l_0, \\ 0 & \text{otherwise,} \end{cases}$$

where l is the distance between the incident masses, l_0 is the rest length of the spring, and k is the spring constant. The second case describes the situation where the cable segment is slack. Slack cable segments do not exert an expanding force along the axis connecting its end points, whence the spring force is set to zero.

The spring constant is related to the material's Young's modulus, E, by the equation

$$k = E\frac{O}{L},$$

where O is the cross-sectional area, and L the length of the entire tether.

Let K be the spring constant of the entire tether and let k_0 be the *unit spring constant*, $k_0 = KL$. Connecting springs in series we have

$$\frac{1}{K} = \sum_{i=1}^{n} \frac{1}{k_i}$$

whence $k_i = k_0/l_{0,i}$ are the spring constants of the cable elements. Assuming all springs to have the same rest length, we obtain that $k = k_0/l_0$.

The damping force acts on the velocity component $v_{\parallel} := \vec{v} \cdot \vec{d}$ in the direction of the vector $\vec{d}$. Its magnitude is given by

$$F_d = -cv_{\parallel},$$

where c is the damping constant. Similarly to the above we have $c = c_0/l_0$, given the damping constant for the entire tether, C, and the *unit damping constant* $c_0 = CL$.

The constants used for our simulation are summarized in Table 4. Whereas the Young's modulus was selected based on data sheets for Dyneema® material, the damping constant was chosen such that the cable model – integrated independently of the kite – could be integrated in a stable fashion. The value of the drag coefficient C_D was taken from [16] for a smooth cylinder at an attack angle of $\pi/2$.

Parameter	Value
Length L	60 m
Diameter	6 mm
Number of point-masses	10
Young's modulus E	55×10^9 Pa
Damping constant C	30×10^3 N $\cdot$ s/m
Drag coefficient C_D	1.22

TABLE 4. Tether parameters.

The top-most node of the tether is synchronized with the location of the kite by the co-simulation master. The aerodynamic forces resulting from the panel method are added onto the tether forces acting on the top node, and these forces together determine the kinematics of this node.

The bottom-most node of the kite is constrained to a fixed position.

REFERENCES

[1] J. Åkesson, M. Gäfvert, and T. Tummescheit. JModelica - an open source platform for optimization of Modelica models. In *Proceedings of MATHMOD 2009 - 6th Vienna International Conference on Mathematical Modelling*, Vienna, 2009.

[2] J. H. Baayen. Vortexje: Source-doublet panel method library. `http://www.baayen-heinz.com/vortexje`, 2012.

[3] J. H. Baayen and W. J. Ockels. Tracking control with adaption of kites. *IET Journal of Control Theory & Applications*, 6(2):182–191, 2012.

[4] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, et al. Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models. In *9th International Modelica Conference*, Munich, 2012.

[5] J. Breukels. *An Engineering Methodology for Kite Design*. PhD thesis, Delft University of Technology, Delft, 2011.

[6] M. Canale, L. Fagiano, and M. Milanese. Power kites for wind energy generation. *IEEE Control Systems Magazine*, 27(6):25–38, 2007.

[7] T. Cebeci. *An engineering approach to the calculation of aerodynamic flows*. Springer, New York, 1st edition, 1999.

[8] A. Deperrois. XFLR5. `http://www.xflr5.com/`, 2012.

[9] K. Dixon, C. S. Ferreira, C. Hofemann, G. van Bussel, and G. van Kuik. A 3d unsteady panel method for vertical axis wind turbines. In *The proceedings of the European Wind Energy Conference & Exhibiti on EWEC Brussels*, pages 1–10, 2008.

[10] L. Dragoş. *Mathematical methods in aerodynamics*. Kluwer Academic Publishers, Dordrecht, 1st edition, 2010.

[11] M. Drela. XFOIL: An analysis and design system for low reynolds number airfoils. Technical report, MIT, 1989.
[12] C. Geuzaine and J.F. Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.
[13] G. Guennebaud, B. Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org`, 2012.
[14] J. L. Hess and A. M. O. Smith. Calculation of potential flow about arbitrary bodies. *Progress in Aeronautical Sciences*, 8:1–138, 1967.
[15] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, 2005.
[16] T. P. Jung. Wind tunnel study of drag of various rope designs. In *28th AIAA Applied Aerodynamics Conference*, San Antonio, 2009.
[17] J. Katz and A. Plotkin. *Low-Speed Aerodynamics*. Cambridge University Press, Cambridge, 2nd edition, 2001.
[18] M. L. Loyd. Crosswind kite power. *Journal of Energy*, 4(3):106–111, 1980.
[19] B. Maskew. Program VSAERO theory document: A computer program for calculating nonlinear aerodynamic characteristics of arbitrary configurations. Technical report, NASA, 1987.
[20] P. Naaijen, V. Koster, and R. P. Dallinga. On the power savings by an auxiliary kite propulsion system. *International shipbuilding progress*, 53(4):255–279, 2006.
[21] R. Rivera, B. Dawes, and D. Abrahams. Boost. C++ libraries. `http://www.boost.org`, 2012.
[22] C. Smith. *On vertex-vertex systems and their use in geometric and biological modelling*. PhD thesis, University of Calgary, Calgary, 2006.
[23] H.A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

E-mail address: `jorn.baayen@baayen-heinz.com`

Jorn H. Baayen, Baayen & Heinz GmbH, Sekr. ER2-1, Hardenbergstr. 36a, 10623 Berlin, Germany.